Session 210

MIDI in Mac OS X

Craig Keithley
USB & FireWire Technology Manager
Apple Worldwide Developer Relations

Session 210

MIDI in Mac OS X

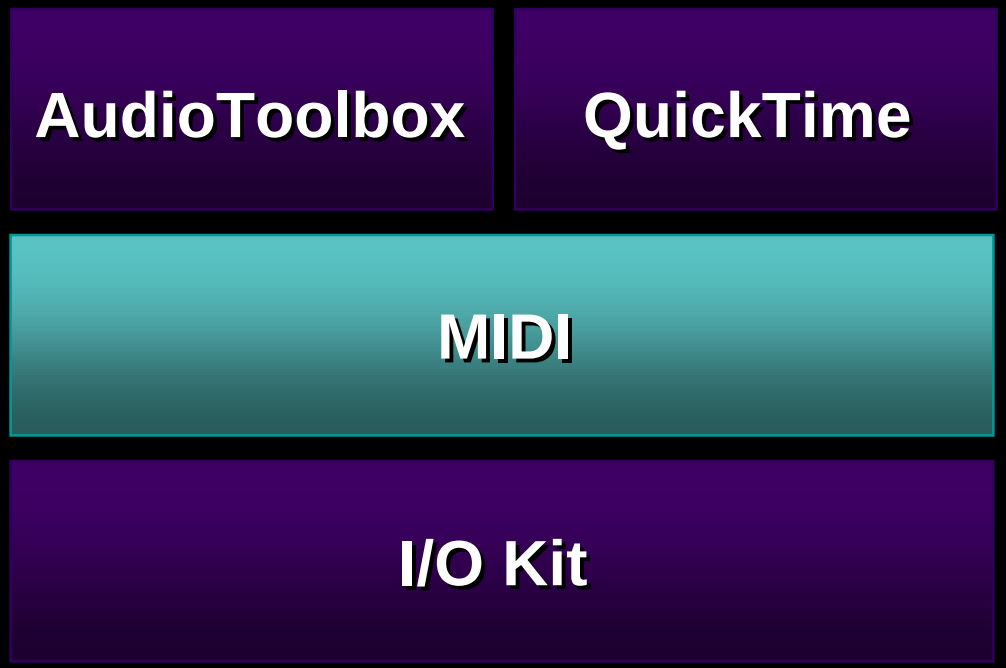
Doug Wyatt
Core Audio

Mac OS X's MIDI System Services

- Introduction
- API concepts/objects
- API usage examples
- Demos

MIDI in the Big Picture

- MIDI services provide high-performance access to MIDI hardware devices



Examples of MIDI Hardware

- USB MIDI interfaces
- USB synthesizers/keyboards
- PCI card
- FireWire MIDI interfaces
- “External” MIDI devices (connect to computer via one of the above, not directly)

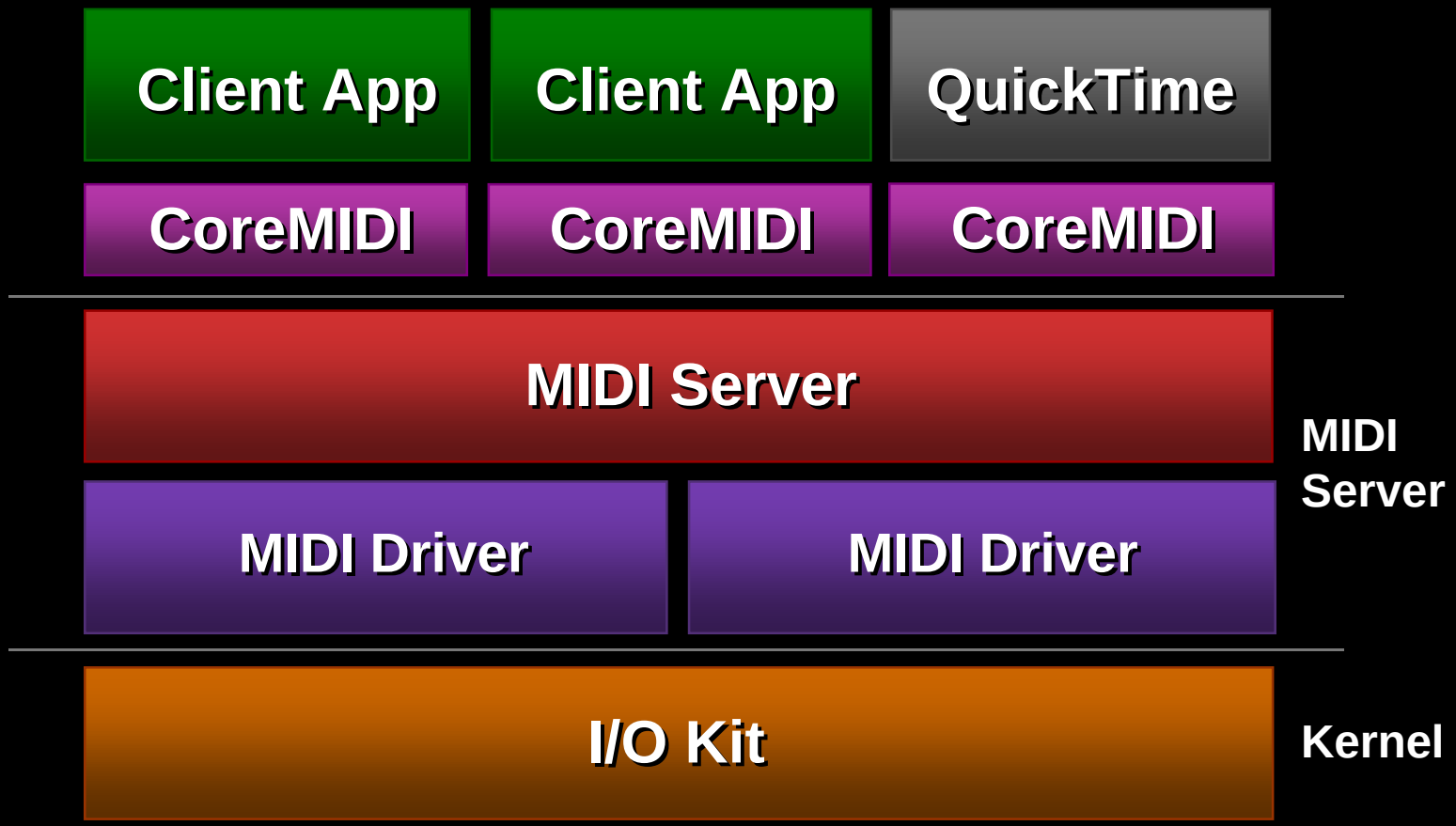
MIDI Services Goals

- Interoperability
- MIDI I/O with highly accurate timing
  - Low latency ($<$ 1 ms thru times)
  - Low jitter (variation in latency) ($<$ 200 μs)
- Single system-wide state
  - Devices, properties
  - Extensible

MIDI Services

- Driver model
- Applications share access to hardware
- Central set of device information
- Timestamped input, scheduled output
- Inter-process communication

MIDI Implementation



MIDI Frameworks

- Mach-O libraries
- CoreMIDI
  - For applications
  - API talks to MIDI Server with Mach messaging
- CoreMIDIServer
  - For drivers (only)
  - Direct access to the API

MIDI Drivers

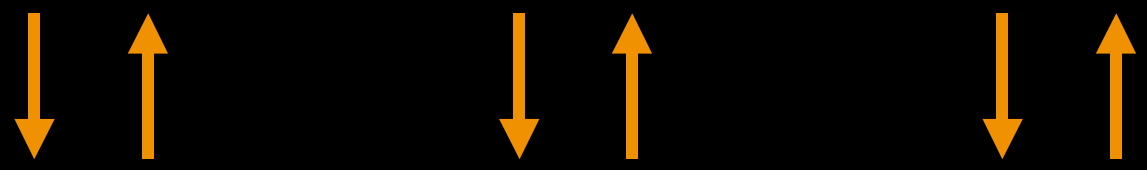
- Apple provides USB MIDI Class standard driver
- Others created by hardware manufacturers
- Installed in/System/Library/Extensions
- Loaded/managed by MIDI Server
- Implemented as user-space CFPlugIn's
- Often, no kernel extension
  ("real" driver) necessary

MIDI Driver Functions

- Simple programming interface
- Detect presence of hardware
  - Create MIDIDevice, MIDIEntity and MIDIEndpoint objects, set their properties
- Perform MIDI I/O, using I/O Kit

MIDIEndpoint

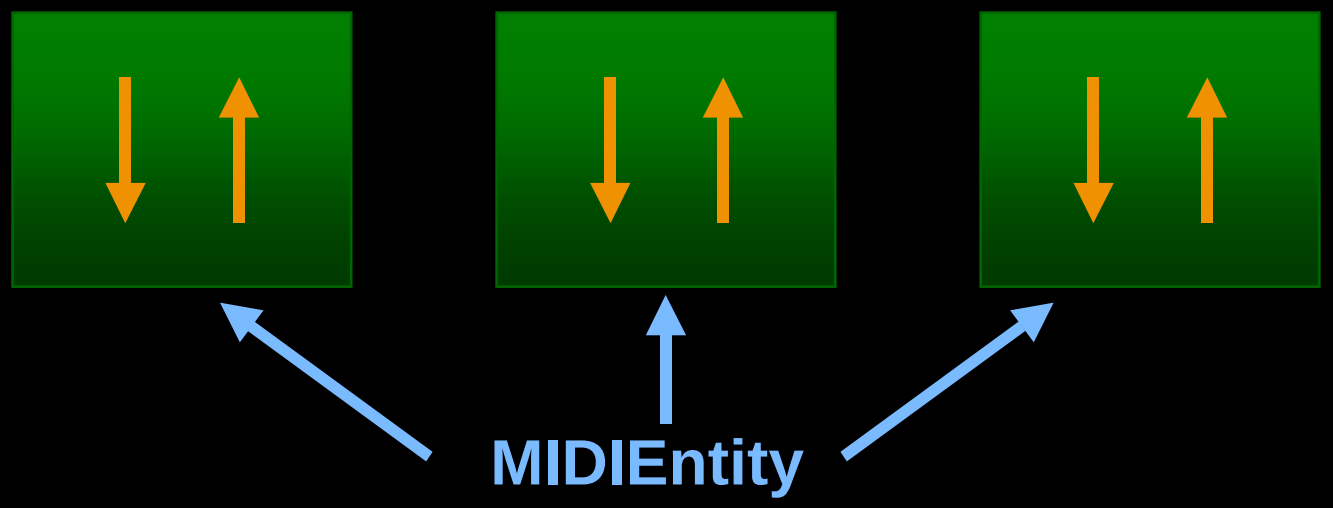
- A MIDI source or destination
- A single 16-channel MIDI stream



MIDIEndpoint

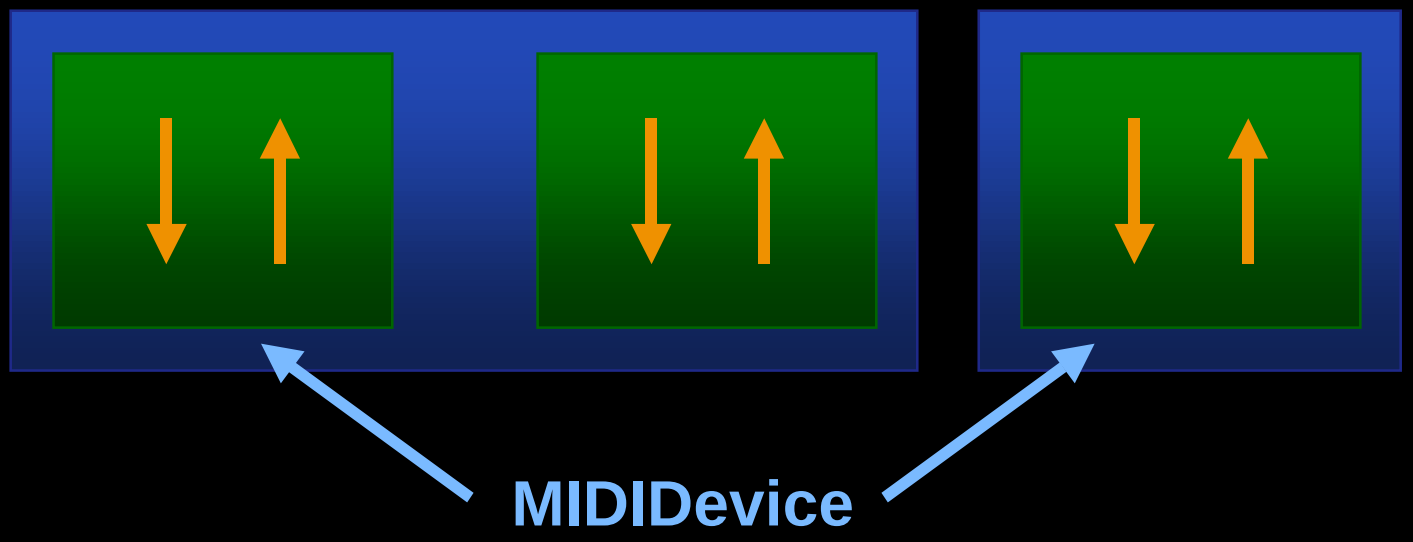
MIDIEntity

- A grouping of endpoints
- A logical subcomponent of a device
- Facilitate bi-directional communication



MIDIDevice

- Represents a physical device
- Created and controlled by a driver
- Contains one or more MIDIEntity objects



MIDIClient

- Before using other MIDI functions, register your client

```
MIDIClientRef client;

MIDIClientCreate(CFSTR("Test client"),
        MyNotifyProc, myRefCon, &client);
```

MIDINotifyProc

- Callback function passed to MIDIClientCreate
- Called in thread which called MIDIClientCreate

```
void MyNotifyProc(const MIDINotification *message, void
*refCon)
{
	switch (message->messageID) {
	case kMIDIMsgSetupChanged:
		// the system state has changed
		break;
	}
}
```

MIDIPort

- Objects through which clients send and receive MIDI
  - (Don’t confuse with MIDI hardware ports)

MIDIPort—Output

- One port may send to all destinations in the system
- Multiple ports: only necessary for separate sending components in an app; MIDIServer will merge them

```
MIDIPortRef outPort;
MIDIOutputPortCreate(client,
    CFSTR("Output port"), &outPort);
```

MIDIPort—Input

- One port may receive input from all sources in the system
- Provides a connection point for input, associated with a MIDIReadProc callback function

```
MIDIPortRef inPort;
MIDIInputPortCreate(client,
        CFSTR("Input port"), MyReadProc, NULL,
        &inPort);
```

MIDIPacketList

- Time-stamped list of packets to or from one endpoint
- All MIDI I/O functions use this structure

```
typedef struct MIDIPacketList {
        UInt32          numPackets;
        MIDIPacket      packet[1];          // open-ended
} MIDIPacketList;
```

MIDIPacket

- Contains one or more simultaneous MIDI events
  - (Exception: system-exclusive events)
  - No running status allowed

```
typedef struct MIDIPacket {
    MIDITimeStamp timeStamp;
    UInt16        length;   // up to 64K
    Byte                    data[256];        // open-ended
} MIDIPacket;
```

Iterating through a MIDIPacketList

```
// wrong: assumes packets are fixed-length
for (int I = 0; I < pktlist->numPackets; ++I) {
        MIDIPacket * pkt = pktlist->packet[I];
      // … work with packet ...
}
// right: aware that packets are variable-length
MIDIPacket *pkt = &pktlist->packet[0];
for (int I = 0; I < pktlist->numPackets; ++I) {
      // … work with packet …
      // use helper macro to get to next packet in list
      pkt = MIDIPacketNext(pkt);
}
```

Building a MIDIPacketList

- Can use helper functions to add messages

```
Byte buffer[1024];
MIDIPacketList *pktlist = (MIDIPacketList *)buffer;

MIDIPacket *curPacket = MIDIPacketListInit(pktlist);

Byte noteOn[] = { 0x90, 60, 64 };

curPacket = MIDIPacketListAdd(pktlist, sizeof(buffer),
curPacket, timeStamp, 3, noteOn);
```

MIDIPacketList Helper Functions

```
// iterate through the packets in a packet list
MIDIPacket *MIDIPacketNext(MIDIPacket *pkt);

// begin building a packet list
MIDIPacket *MIDIPacketListInit(MIDIPacketList *pktlist);

// add an event to a packet list
MIDIPacket *MIDIPacketListAdd(MIDIPacketList *pktlist, ByteCount
    listSize, MIDIPacket *curPacket, MIDITimeStamp time, ByteCount nData,
    Byte *data);
```

Finding and Sending to Destinations

```
ItemCount nDests = MIDIGetNumberOfDestinations();

for (int iDest = 0; iDest < nDests; ++iDest) {

    MIDIEndpointRef dest = MIDIGetDestination(iDest);

    MIDISend(gOutputPort, dest, &packetList);
}
```

Finding and Receiving From Sources

```
ItemCount nSrcs = MIDIGetNumberOfSources();
for (int iSrc = 0; iSrc < nSrcs; ++iSrc) {
    MIDIEndpointRef src = MIDIGetSource(iSrc);
    void *srcConnRefCon = src;
    MIDIPortConnectSource(inPort, src,
            srcConnRefCon );
}

void MyReadProc(const MIDIPacketList *pktlist,
    void *readProcRefCon, void *srcConnRefCon);
```

MIDIReadProc

- Callback function to receive MIDI input
- Called in high-priority thread created by CoreMIDI.framework

```
void MyReadProc(const MIDIPacketList *pktlist, void
*procRefCon, void *srcRefCon)
{
	// …process packet list ...
}
```

Iterating Through Devices and Entities

```
ItemCount ndev = MIDIGetNumberOfDevices();
for (int idev = 0; idev < ndev; ++idev) {
        MIDIDeviceRef device = MIDIGetDevice(idev);
        ItemCount nent =
                MIDIGetNumberOfEntities(device);
                for (int ient = 0; ient < nent; ++ient) {
                MIDIEntityRef entity =
                MIDIGetEntity(device, ient);
            ...
        }
}
```

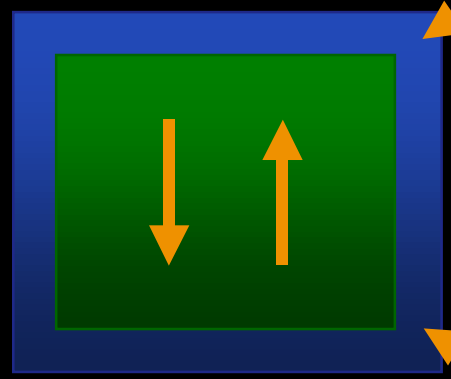
Devices or Endpoints?

- Iterating through endpoints
  - Excludes endpoints of devices which are offline
  - Includes virtual endpoints
  - Use for I/O purposes
- Iterating through devices
  - Includes devices which are offline
  - Won't locate virtual endpoints
  - Use for configuration purposes

Properties

- Devices, entities, endpoints have them
- Extensible
  - Drivers set them
  - Applications typically just read them, but may add their own

Property Inheritence



Device
Name = X999
Manufacturer = Xcorp
Model = X999

Entity
Name = Port1
Manufacturer = Xcorp
Model = X999
SysExID = 17

Endpoint
Name = Port1 In
Manufacturer = Xcorp
Model = X999
SysExID = 17

Common Properties

- Name
- Device's manufacturer and model names and system-exclusive ID
- Device's maximum transmit speed (sys-ex)
- Device's schedule-ahead time

String Property Example

```
CFStringRef cfName;
char name[64];
MIDIObjectGetStringProperty(device,
        kMIDIPropertyName, &cfName);
CFStringGetCString(cfName,
        name, sizeof(name), 0);
printf("The device's name is %s\n", name);
CFRelease(cfName);
```

Numeric Property Example

```
SInt32 time;
MIDIObjectGetIntegerProperty(dest,
        kMIDIPropertyAdvanceScheduleTimeMuSec,
        &time);
printf("The destination's advance scheduling time is %ld µs\n",
(long)time);
```

Using Timestamps

```
#include <CoreAudio/HostTime.h>

MIDITimeStamp hostTime =
        AudioGetCurrentHostTime();

UInt64 nanoseconds = AudioConvertHostTimeToNanos(hostTime);

MIDITimeStamp backToHostTime =
        AudioConvertNanosToHostTime(nanoseconds);
```

Scheduling MIDI Output

- Schedule future output using MIDITimeStamps (host time)
- Highly accurate scheduler; uses Mach realtime-priority thread
- Don't schedule too far in advance (> ~100ms)—you can't unschedule
- Drivers' devices may have a property for minimum advance scheduling time—check it

Other Timing Issues

- Incoming MIDI is timestamped with a host clock
- To schedule your own tasks, use Carbon's Multiprocessing services

Strategies for Efficiency

- When possible, schedule output at least a few milliseconds in advance
- Send multiple MIDI events that happen close together in time with a single call to MIDISend
- Strategies are not required—just helps you be a better citizen

Performance

- MIDI Thru time: < 1 ms
- Scheduler wakeup jitter: $< 100\ \mu s$

Interprocess Communication

- Your app can create virtual sources and/or destinations
- All clients (including your app) will see the virtual endpoints as if they were ordinary sources and destinations

Virtual Source Example

```
// create a virtual source
MIDIEndpointRef virtualSource;
OSStatus err = MIDISourceCreate(myClient, CFSTR("my virtual source"),
        &virtualSource);

// send a MIDIPacketList to any clients connected to a virtual source
err = MIDIReceived(virtualSource,
        packetList);
```

Virtual Destination Example

```
// create a virtual destination
MIDIEndpointRef virtualDest;
OSStatus err = MIDIDestinationCreate(myClient, CFSTR("my virtual
        dest"), MyReadProc, aReadProcRefCon, &virtualDest);

// MyReadProc will get called with incoming MIDI data when any client
        sends to "my virtual dest"
```

Sending System-Exclusive Messages

- (e.g., patch librarian apps)
- Need to “meter” large outgoing MIDI messages so as to avoid overflows
  - Check kMIDIPropertyMaxSysExSpeed
  - Or use MIDISendSysex

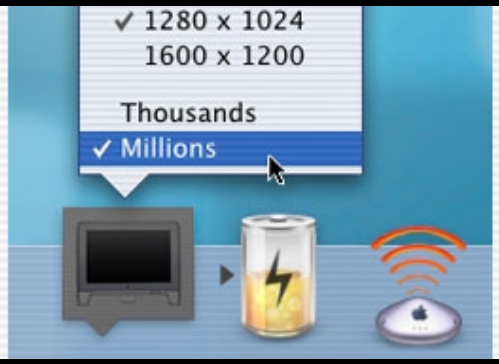
MIDISendSysex Example

```
void SendSysex(MIDIEndpointRef dest, Byte *sysexMessage, UInt32
        messageLength)
{
        MIDISysexSendRequest req;
        req.destination = dest;
        req.data = sysexMessage;
        req.bytesToSend = messageLength;
        req.completionProc = MyCompletionProc;
        req.completionRefCon = NULL;
        OSStatus err = MIDISendSysex(&req);
        // await completion
        while (!req.complete) sleep(1);
}
```

MIDISendSysex Notes

- Make sure the request structure is not deal located before the request completes
- You can abort a request by setting complete to true
- Runs in its own medium-priority thread, within your app

Demo



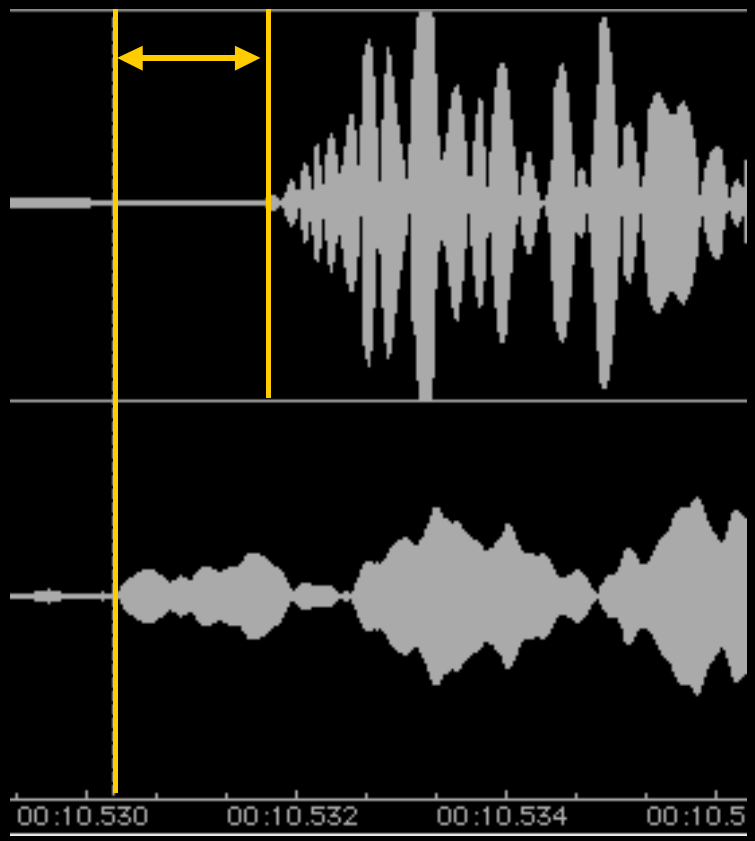
MIDI Thru Performance

Play note on keyboard…

1-2 ms

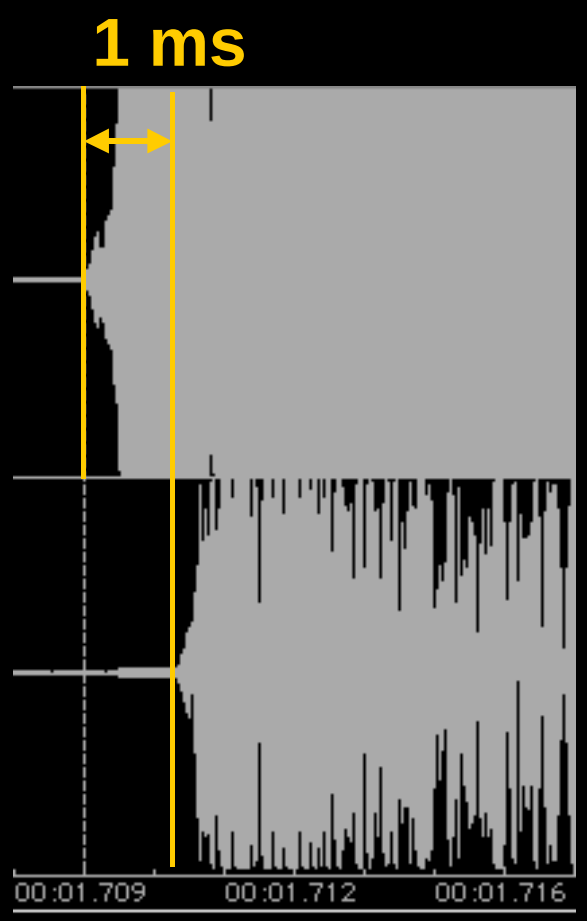
Keyboard -> Mac -> Device MIDI Ch 1 via USB, MIDI Server, MIDI thru app

Local control, MIDI Ch 2



MIDI/Audio Performance

From a single MIDI trigger on the Mac...



Using MIDI in Your App

- Available in 10.0
- Documentation
  - In the framework header files
    - CoreMIDI.framework
    - CoreMIDIServer.framework
- Examples
  - /Developer/Examples/CoreAudio/MIDI

CoreMIDI Java

Bill Stewart
Core Audio

Demo

MIDI Graph

Core MIDI Java

- Java Doc available for API
- Should understand the fundamental designs of core MIDI framework
  - Regardless of language used
- Java API presents the same functionality

Resources

Core Audio Mailing List
Specifications and SDKs
http://lists.apple.com

Developer Web Site
http://developer.apple.com/audio

Related Sessions

136 Sound & Networking for Games: Enabling games with audio & networking	Room J2 **Fri., 2:00pm**
209 Audio Processing & Sequencing Services: Using Core Audio in Mac OS X	Room J2 **Fri., 3:30pm**
208 Audio Services in Mac OS X: Using Core Audio in Mac OS X	Room J2 **Thur., 5:00pm**

Who to Contact

Apple Worldwide Developer Relations

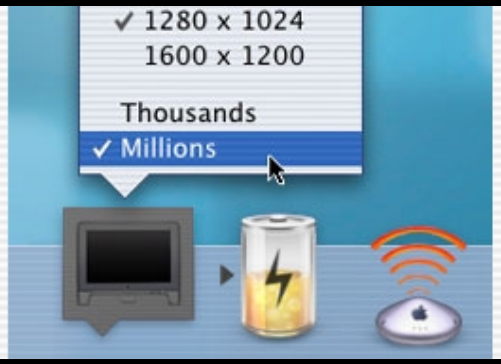
Craig Keithley
FireWire & USB Technology Manager
keithley@apple.com

Developer Seeding

audio@apple.com

http://developer.apple.com/events/urls.html

Q&A



Craig Keithley
FireWire & USB Technology Manager
keithley@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001